```
// C++ program to solve knapsack problem using
branch and bound
#include <bits/stdc++.h>
using namespace std;
struct Item
{
    float weight;
    int value;
};
struct Node
{
    int level, profit, bound;
    float weight;
};
bool cmp(Item a, Item b)
{
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}
int bound(Node u, int n, int W, Item arr[])
{
    if (u.weight >= W)
        return 0;
    int profit_bound = u.profit;
    int j = u.level + 1;
    int totweight = u.weight;
    while ((j < n) && (totweight + arr[j].weight <= W))
    {
        totweight += arr[j].weight;
        profit_bound += arr[j].value;
        j++;
    }
    if (j < n)
        profit_bound += (W - totweight) * arr[j].value /
                                         arr[j].weight;
    return profit_bound;
}
{
    sort(arr, arr + n, cmp);
    queue<Node> Q;
    Node u, v;
    u.level = -1;
    u.profit = u.weight = 0;
    Q.push(u);
    int maxProfit = 0;
    while (!Q.empty())
    {
        u = Q.front();
        Q.pop();
        if (u.level == -1)
            v.level = 0;
        if (u.level == n-1)
            continue;
        v.level = u.level + 1;
        v.weight = u.weight + arr[v.level].weight;
        v.profit = u.profit + arr[v.level].value;
        if (v.weight <= W && v.profit > maxProfit)
            maxProfit = v.profit;
        v.bound = bound(v, n, W, arr);
            if (v.bound > maxProfit)
            Q.push(v);
        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = bound(v, n, W, arr);
        if (v.bound > maxProfit)
            Q.push(v);
    }
    return maxProfit;
}
int main()
{
    int W = 10; // Weight of knapsack
    Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100},
                {5, 95}, {3, 30}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum possible profit = "
        << knapsack(W, arr, n);
    return 0;
```

```
}
```

📥 **Input:**

W = 10; Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100}, {5, 95}, {3, 30}};

📤 **Output:**

Maximum possible profit = 235

```
Maximum possible profit = 235
```

Time Complexity: O(2N)

Auxiliary Space: O(N)